

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по практической работе №4**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Деревья**

Студент гр. 8304

Преподаватель

Ястребов И.М.

Фирсов М.А.

Санкт-Петербург 2019

## Цель работы

Научиться реализовать рандомизированное бинарное дерево поиска (РБДП) и основные функции работы с ним.

## Задание

Рассматриваются бинарные деревья с элементами типа Elem (в качестве Elem использовать char). Заданы перечисления узлов некоторого дерева b в порядке КЛП и ЛКП. Требуется:

- восстановить дерево b и вывести его изображение;
- перечислить узлы дерева b в порядке ЛПК.

## Описание алгоритма

1. Открывается файл с тремя входными строками.
2. Первая определяет вид представления графа во входной строке.
3. Вторая — дерево в строчной записи
4. Третья — вид представления графа в выходной строке.
5. Узлы графа создаются соответственно способу представления графа в строке
6. При выводе используется соответствующий способ обхода — ЛПК, КЛП или ЛКП.

Тестирование программы приведено в Приложении А, исходный код программы представлен в Приложении Б.

## Описание основных функций

1. `sIter` bracket\_closer(`sIter` begin)

Функция находит итератор соответствующей парной скобки для данной

- 2.

```
nodePtr<Elem> readTreeFromStringNLR(std::string&, sIter, sIter);
nodePtr<Elem> readTreeFromStringLNR(std::string&, sIter, sIter);
nodePtr<Elem> readTreeFromStringLRN(std::string&, sIter, sIter);
void printTreeNLR(nodePtr<Elem> root);
void printTreeLNR(nodePtr<Elem> root);
void printTreeLRN(nodePtr<Elem> root);
```

Функции производят считывание и вывод графа в соответствующем виде.

**Вывод.**

Были получены навыки работы с деревьями, структура данных была реализована на языке программирования C++.

## ПРИЛОЖЕНИЕ А

### Тестирование программы

```
test#0
input in NLR
input 8(3(1##)(6(4##)(7##)))(10#(14(13##)#))
output in LNR
(((#1#)3((#4#)6(#7#)))8(#10((#13#)14#)))
```

14

13

10

8

7

6

4

3

1

```
test#1
```

## ПРИЛОЖЕНИЕ Б

### Файл Lab4.cpp

```
#include "pch.h"

#include <iostream>
#include <fstream>
#include "BinTree.hpp"

int main(int argc, char* argv[])
{
    BinTree<std::string> bTree;
    std::string input;
    int test_cnt = 0;

    if (argc == 1)
    {
        std::ifstream iStream("test.txt");
        std::string mode;

        while (std::getline(iStream, mode)) {

            std::cout << "_____\\ntest#" << test_cnt++ <<
std::endl;

            std::cout << "input in " << mode << std::endl;

            std::getline(iStream, input);

            std::cout << "input " << input << std::endl;

            if (mode == "NLR")
                bTree.head = bTree.readTreeFromStringNLR(input,
input.begin(), input.end());

            if (mode == "LNR")
                bTree.head = bTree.readTreeFromStringLNR(input,
input.begin(), input.end());

            if (mode == "LRN")
                bTree.head = bTree.readTreeFromStringLRN(input,
input.begin(), input.end());

            std::getline(iStream, mode);

            if (mode == "NLR")
            {
                std::cout << "output in " << mode << std::endl;
                bTree.printTreeNLR(bTree.head);
            }

            if (mode == "LNR") {
                std::cout << "output in " << mode << std::endl;
                bTree.printTreeLNR(bTree.head);
            }

            if (mode == "LRN")
            {
                std::cout << "output in " << mode << std::endl;
```

```

        bTree.printTreeLRN(bTree.head);
    }

    std::cout << std::endl;

    print2D(bTree.head);
}

if (argc == 2)
{
    std::ifstream iStream;

    std::string mode;

    iStream.open(argv[1]);

    while (std::getline(iStream, mode))
    {
        std::cout << "test#" << test_cnt++ << std::endl;

        std::cout << "input in " << mode << std::endl;

        std::getline(iStream, input);

        std::cout << "input " << input << std::endl;

        if (mode == "NLR")
            bTree.head = bTree.readTreeFromStringNLR(input,
input.begin(), input.end());

        if (mode == "LNR")
            bTree.head = bTree.readTreeFromStringLNR(input,
input.begin(), input.end());

        if (mode == "LRN")
            bTree.head = bTree.readTreeFromStringLRN(input,
input.begin(), input.end());

        std::getline(std::cin, mode);

        if (mode == "NLR")
        {
            std::cout << "output in " << mode << std::endl;
            bTree.printTreeNLR(bTree.head);
        }

        if (mode == "LNR")
        {
            std::cout << "output in " << mode << std::endl;
            bTree.printTreeLNR(bTree.head);
        }

        if (mode == "LRN")
        {
            std::cout << "output in " << mode << std::endl;
            bTree.printTreeLRN(bTree.head);
        }

        std::cout << std::endl;
    }
}
}

```

## Файл BinTree.hpp

```
#pragma once
#include <variant>
#include <memory>
#include <string>

typedef std::string::iterator sIter;

template <typename Elem>
class Node;

template <typename Elem>
class BinTree;

template<typename Elem>
using nodePtr = std::shared_ptr<Node<Elem>>;

#define COUNT 10

template<typename Elem>
void print2DUtil(nodePtr<Elem> root, int space)
{
    // Base case
    if (root == NULL)
        return;

    // Increase distance between levels
    space += COUNT;

    // Process right child first
    print2DUtil(root->right, space);

    // Print current node after space
    // count
    std::cout << std::endl;
    for (int i = COUNT; i < space; i++)
        std::cout << " ";
    std::cout << root->value << "\n";

    // Process left child
    print2DUtil(root->left, space);
}

// Wrapper over print2DUtil()
template<typename Elem>
void print2D(nodePtr<Elem> root)
{
    // Pass initial space count as 0
    print2DUtil(root, 0);
}

sIter bracket_closer(sIter begin)
{
    size_t tmp = 1;

    while (tmp)
    {
        begin++;
    }
}
```

```

        if (*begin == '(')
            tmp++;

        if (*begin == ')')
            tmp--;
    }

    return begin;
}

template <typename Elem>
class Node
{
public:
    Node(Elem value) {
        this->value = value;
    }

    Node(const Node<Elem> &copy) {
        left = std::make_shared<Node<Elem>>();
        *left = *(copy.left);

        right = std::make_shared<Node<Elem>>();
        *right = *(copy.right);

        value = copy.value;
    }

    Node<Elem>& operator=(const Node<Elem> &copy)
    {
        left = std::make_shared<Node<Elem>>();
        *left = *(copy.left);

        right = std::make_shared<Node<Elem>>();
        *right = *(copy.right);

        value = copy.value;

        return *this;
    }

    Node() = default;

    ~Node() = default;

    Elem value;

    nodePtr<Elem> left;
    nodePtr<Elem> right;
};

```

```

template<typename Elem>
class BinTree
{
public:

    BinTree() = default;
    ~BinTree() = default;

```



```

BinTree(const BinTree<Elem> &copy) {
    head = std::make_shared<Node<Elem>>();

    *head = *(copy.head);
}

BinTree<Elem>& operator=(const BinTree<Elem> &copy) {
    head = copy.head;

    *head = *(copy.head);

    return *this;
}

nodePtr<Elem> head;

nodePtr<Elem> readTreeFromStringNLR(std::string&, sIter, sIter);
nodePtr<Elem> readTreeFromStringLNR(std::string&, sIter, sIter);
nodePtr<Elem> readTreeFromStringLRN(std::string&, sIter, sIter);

void printTreeNLR(nodePtr<Elem> root);
void printTreeLNR(nodePtr<Elem> root);
void printTreeLRN(nodePtr<Elem> root);
};

template<typename Elem>
nodePtr<Elem> BinTree<Elem>::readTreeFromStringNLR(std::string &source,
sIter begin, sIter end)
{
    std::string root;

    sIter tmp = begin;

    while ((*tmp != '(') && (*tmp != '#') && (*tmp != ')'))
        root += *(tmp++);

    auto res = std::make_shared<Node<Elem>>();
    res->value = (Elem)root;

    if (*tmp == ')') {
        res->left = nullptr;
        res->right = nullptr;
        return res;
    }

    if (*tmp == '#')
        res->left = nullptr;

    else {
        sIter tmpCopy = tmp;
        tmp = bracket_closer(tmpCopy);
        res->left = readTreeFromStringNLR(source, tmpCopy + 1, tmp - 1);
    }

    ++tmp;

    if (*tmp == '#')
        res->right = nullptr;

    else
    {

```

```

        res->right = readTreeFromStringNLR(source, tmp + 1,
bracket_closer(tmp) - 1);
    }

    return res;
}

template<typename Elem>
nodePtr<Elem> BinTree<Elem>::readTreeFromStringLNR(std::string &source,
sIter begin, sIter end)
{
    auto res = std::make_shared<Node<Elem>>();

    sIter tmp = begin;

    if (*tmp == '#')
        res->left = nullptr;

    else {
        sIter tmpCopy = tmp;
        tmp = bracket_closer(tmpCopy);

        res->left = readTreeFromStringLNR(source, tmpCopy + 1, tmp - 1);
    }

    ++tmp;

    std::string root;

    while ((*tmp != '(') && (*tmp != '#'))
        root += *(tmp++);

    res->value = (Elem)root;

    if (*tmp == '#') {
        res->right = nullptr;
    }

    else {
        res->right = readTreeFromStringLNR(source, tmp + 1,
bracket_closer(tmp) - 1);
    }

    return res;
}

template<typename Elem>
inline nodePtr<Elem> BinTree<Elem>::readTreeFromStringLRN(std::string
&source, sIter begin, sIter end)
{
    sIter tmp = begin;

    auto res = std::make_shared<Node<Elem>>();

    if (*tmp == '#')
        res->left = nullptr;

    else {
        sIter tmpCopy = tmp;
        tmp = bracket_closer(tmpCopy);

        res->left = readTreeFromStringLRN(source, tmpCopy + 1, tmp - 1);
    }
}

```

```

    ++tmp;

    if (*tmp == '#')
        res->right = nullptr;

    else
    {
        sIter tmpCopy = tmp;
        tmp = bracket_closer(tmpCopy);
        res->right = readTreeFromStringLRN(source, tmpCopy + 1, tmp - 1);
    }

    ++tmp;

    std::string root;

    while ((tmp <= end) && (tmp < source.end()))
        root += *(tmp++);

    res->value = (Elem)root;

    return res;
}

template<typename Elem>
void BinTree<Elem>::printTreeLRN(nodePtr<Elem> root)
{
    if (!root) {
        std::cout << '#';
        return;
    }

    std::cout << '(';

    printTreeLRN(root->left);
    printTreeLRN(root->right);

    std::cout << root->value << ')';
}

template<typename Elem>
void BinTree<Elem>::printTreeLNR(nodePtr<Elem> root)
{
    if (!root) {
        std::cout << '#';
        return;
    }

    std::cout << '(';

    printTreeLNR(root->left);
    std::cout << root->value;
    printTreeLNR(root->right);

    std::cout << ')';
}

template<typename Elem>
void BinTree<Elem>::printTreeNLR(nodePtr<Elem> root)
{
    if (!root) {
        std::cout << '#';

```

```
        return;
    }

    std::cout << '(';

    std::cout << root->value;

    printTreeNLR(root->left);
    printTreeNLR(root->right);

    std::cout << ')';
}
```